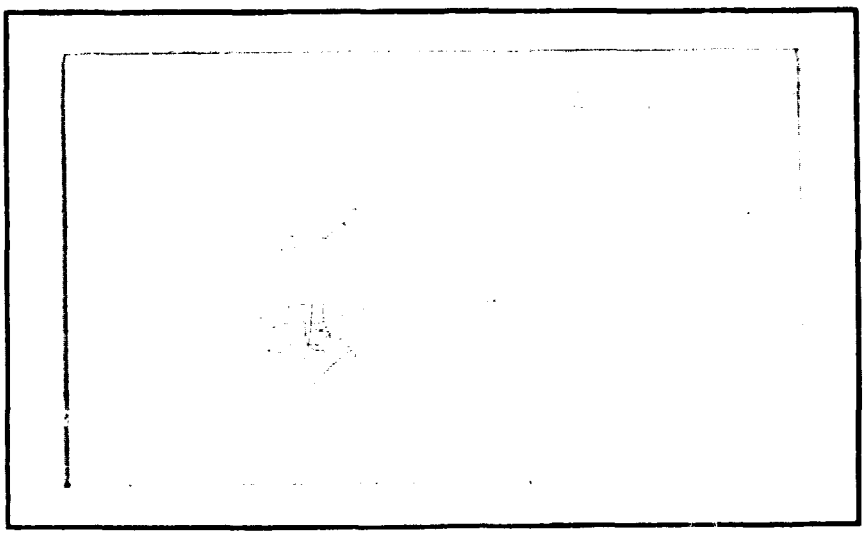
CR-171898
C.1



UNIVERSITY of PENNSYLVANIA

The Moore School of Electrical Engineering

PHILADELPHIA, PENNSYLVANIA 19104

User Interface Enhancement Report

May 31, 1985

NASA Contract No. NAS9-16634

Norman I. Badler,
Principal Investigator

Jeffrey Gangel
Gwen Shields
Glenn Fala

Department of Computer and Information Science
Moore School D2
University of Pennsylvania
Philadelphia, PA 19104

Table of Contents

List of Figures

List of Tables

1. Introduction

The existing user interfaces to TEMPUS, Plaid, and other systems in the OSDS are fundamentally based on only two modes of communication: alphanumeric commands or data input and graphical interaction. The latter are especially suited to the types of interaction necessary for creating workstation objects with BUILD and with performing body positioning in TEMPUS. Looking toward the future application of TEMPUS, however, we see that the long-term goals of OSDS will include the analysis of extensive *tasks* in space involving one or more individuals working in concert over a period of time. In this context, the TEMPUS body positioning capability, though extremely useful in creating and validating a small number of particular body positions, will become somewhat tedious to use. The *macro* facility helps somewhat, since frequently used positions may be easily applied by executing a stored macro. The difference between body positioning and task execution, though subtle, is important. In the case of task execution, the important information at the *user's* level is *what actions are to be performed* rather than *how the actions are performed.* Viewed slightly differently, the *what* is constant over a set of individuals though the *how* may vary.

To meet these future expected needs of OSDS we began a study of more language-oriented interfaces to TEMPUS. Language can describe how to perform an action, but it is much more efficient and effective in describing what is to be accomplished. Human intelligence, innate movement patterns, learned actions, and general experience make possible human task performance. TEMPUS attempts to simulate at least some of these, such as reach. A language-oriented interface would permit the TEMPUS user to describe the steps and goals of a task in a form much like, or even identical to, the form *interpretable by another human being.* There is no doubt that this is a modern area of extensive research covered even more broadly by the field of Artificial Intelligence. Thus our purpose will not be to attempt the analysis of arbitrary human communication; rather, we shall concentrate on those areas of physical task execution that are fundamental to the NASA environment.

We begin by describing some of the characteristics of a language-oriented user interface, examine some of the representational problems that arise, and consider how they might be solved. Then two common NASA task specification techniques, operational callouts and commands, are reviewed, analyzed, and represented. Examples are used extensively to illustrate both kinds of task instruction and how they may be parser and understood by computer. Finally we show how the structures representing

these language inputs are provided to TEMPUS and its associated movement simulation system.

2. A Language-Oriented User Interface

A problem with many systems is the manner in which the input must be formatted. Often, a user would like to communicate with a system in his or her own language, for example English. In practice, commmunication with a computer through arbitrary (but humanly understandable) English is not possible. Thus we frequently resort to more limited syntactic, semantic, and lexical domains. Such an interface appears to understand a restricted form of English utterances that nonetheless transmits most of the required information to the computer.

In the case of TEMPUS the interface must allow the user to command the (simulated) agents to perform a number of tasks. If the interface is treated as a participant in a conversation (so to speak), then everything entered should be understood by the interface and conveyed to the system. In TEMPUS, however, the interface must do more. The simulated agents have no action intelligence and require a specific type of statement to allow them to perform in the manner that was implied. By allowing the interface to be *omniscient*, at least in the context of a specific environment, the message sent can be conveyed, with all the information needed by the simulator. Thus the interface is an intermediary between the TEMPUS user and available simulation capabilities, with (relatively) complete insulation of the user from the details of the required simulation structure.

The interface between the user and the simulation system accepts statements presented either in isolation by the operator or from stored files of task descriptions. In either case, all statements pertinent to this discussion are entered in a command form, since that is exactly what we would want to do: command agents to perform some actions. The interface then tries to comprehend the command and determines what, if any, additional information may be needed from the user. The information available to it, some from prior information, some from the command, some from the user (if needed), and much from its knowledge about the objects in the environment, is then changed into a format compatible with the simulator and then sent to it. To allow such a system to work, what is first needed is an understanding of the underlying meaning of the command.

There are two types of commands we will examine which are stereotypical of a wide class of action instructions found in NASA checklist procedure documents (*e.g.* [5]). The first class of commands consist of tersely abbreviated *operational callouts*; the second of more language-like *commands*. We will examine these two classes in this order as the problems in maintaining a computer-understandable representation of the task information embodied in them is easier to study in the more standardized operational callouts. The more general commands lead into deeper natural language parsing and comprehension issues.

3. Processing NASA Checklists for TEMPUS Input

To improve the user interface to the TEMPUS body positioning system, the use of existing NASA checklist procedures was investigated. This section begins by discussing a computer-oriented representation for the kind of information contained in NASA checklist procedures. We present a structure, based on a representational frame, that will accurately represent these procedures. The task representation will consist of action verb case frames acting on a database of object frames. Many examples are given to aid in describing how the checklist procedures can be parsed into the chosen task representation. The parser will interpret as much information as possible from the checklist procedures, and will relay this information in detail to a simulation system [3] which drives the TEMPUS body motion primitives. Finally, areas of known or potential difficulty are noted.

3.1. Task Representation Requirements

NASA checklist procedures are a complicated system of abbreviations and artificial syntax which is nonetheless intended to be human-readable. They represent all actions to be carried out by the crewmembers (called, in general, *agents*) from the time the shuttle is entered until it is exited. Our goal is to attempt to interface between the NASA checklist procedures (what is to be done) and the TEMPUS graphic simulation (what will it look like). This is done through an intermediate structure called a *task representation*. The task representation must be capable of specifying parallel sequences of actions. It must not be too detailed so that it must be changed for a different (size) agent, changed location of an agent, or different physical arrangement of the workstation. The task representation must be able to make explicit which agent or agents carry out which task, with which of their resources, and under what conditions.

The task specification will be compiled first by translating the NASA checklist procedures into a task representation. The checklist procedures are in the form of terse NASA documents called the Flight Data File (FDF). The FDF describes the tasks to be carried out by the crewmembers of the Space Shuttle. The task representation is passed to a simulation which decides how each action will be carried out.

Several problems are involved with designing the task representation. Of major concern is the representation of temporal relationships between actions. It is essential that each action be done at exactly the time and order designated. NASA documents spend a large amount of effort describing and representing the temporal relationships of the actions to be performed. Allen [1] discusses various temporal relationships of actions and how to represent them. He addresses three problems that he feels occur in nearly all existing (computationally-based) models of action.

1. Actions that involve non-activity.
2. Actions that are not easily decomposed into sub-actions.
3. Actions that occur simultaneously and may interact with each other.

The third point is of particular interest because of the importance of accurately representing sequences of actions, parallel actions, and their interactions. Allen concludes that simultaneous actions can be described directly if the temporal aspects are separated from the causal aspects of a plan and from that it enables us to describe the interactions as well. He feels that problems still remain in building a system that can *reason* about such interactions while problem solving, but the representation appears suitable for our simulation needs. Allen describes thirteen possible relationships, summarized in Figure 3-1.

More generally, we must address the problem of representing the entire procedure or event. Schank [9] proposed a *Conceptual Dependency grammar* (CD) for representing events using several basic primitives for action verbs. His theory says that all actions can be reduced to a sequence of these primitives and he bases his theory on the idea of a case grammar. He chooses relevant slots to represent the events. His work has several weaknesses but the ideas of a *case frame*, relevant slots and primitive verbs will be the backbone of the task representation, which will be described in Section 3.3. For our purposes, it is sufficient to understand that a case frame is a list of attributes (cases or slots) of a verb into which varying types of information (values) may be placed.

Table 3-1: Allen's Temporal Primitives

Symbol	Temporal Primitive	Example
<	X before Y	XXX YYY
>	X after Y	YYY XXX
=	X equal Y	XXX YYY
m	X meets Y	XXXYYY
mi	X is met by Y	YYYXXX
o	X overlaps Y	XXX YYY
oi	X is overlapped by Y	YYY XXX
d	X during Y	XXX YYYYYY
di	X includes Y	XXXXXX YYY
s	X starts Y	XXX YYYYYY
si	X is started by Y	XXXXXX YYY
f	X finishes Y	XXX YYYYY
fi	X is finished by Y	XXXXX YYY

Case grammar argues that a set of cases dependent on the verb can be used to define the deep structure ("meaning," as opposed to the superficial sentence structure) of English sentences. Schank's CD representation system is based on the conceptual relationships between objects and actions. As in all case grammar structures, the meaning of a sentence is not dependent on the form of the sentence. The sentences

```
Jeff turned the valve
```

and

```
The valve was turned by Jeff.
```

both have the same deep structure. Schank's CD extends the case grammar concept one step forward and reduces all verbs into a member of a set of limited primitive verbs (Table 3-2). As a result, both of the above sentences would be reduced into the case frame:

```
EVENT
   actor:  Jeff
   action: PROPEL
   object: the valve.
```

Case frames contain a set of roles and the constituents that fill them. The roles in the above example, ACTOR, ACTION, and OBJECT are filled with their constituents, *Jeff*, *PROPEL*, and *the valve*.

Table 3-2: Schank's Primitive Action Verbs

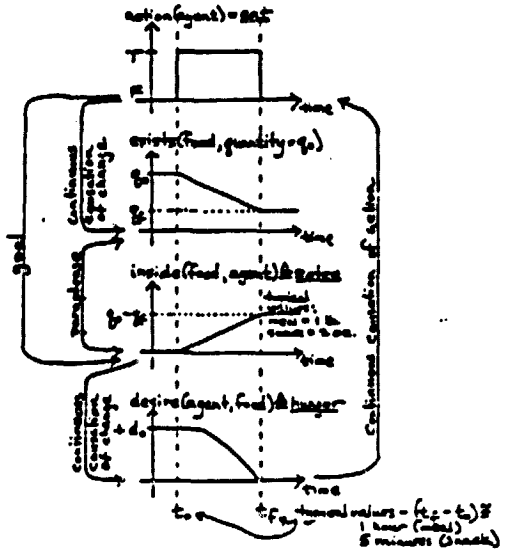
Primitive Verb	Meaning	Sentence	CD Example
PTRANS	physical transfer	John moves to Hawaii.	John PTRANS Hawaii
ATRANS	abstract transfer	John buys a book from the store.	John ATRANS book Store ATRANS money
MTRANS	mental transfer	John reads a book.	John MTRANS book
PROPEL	apply physical force	John throws a ball.	John PROPEL ball
MBUILD	mental operation (old info to new info)	John understands the formula.	John MBUILD formula
ATTEND	focus attention	John hears a new song.	John MBUILD new song John ATTEND new song
GRASP	gain physical control (with your hands)	John holds a pencil.	John GRASP pencil
MOVE	bodily movement	John drinks coffee.	John MOVE coffee

Only recently have efforts been made to construct a representation combining the conceptual and the temporal. Besides the efforts cited in a companion report [3], relevant work in this area was done by Waltz [10]. Waltz addresses some of the shortcomings of Schank's Conceptual Dependency representation system, one of which is capturing the similarity of meaning of various verbs rather than representing the nuances in meaning. In a Space Shuttle environment each meaning must be interpreted exactly right or the wrong action may be performed. Waltz also feels that there are no primitives in Schank's representation for many actions. Waltz's *Event Shape Diagrams* deal with action combinations and result in a temporal framework. Although his paper is short and just scratches the surface of this concept, it offers useful ideas in setting up the task representation. The event shape diagrams can be used to represent concurrent processes, causation and other temporal relations by aligning two or more diagrams[1] (see Figure

[1]Its similarity to the TEMPUS track-oriented animation system is worth noting [4].

3-1).

Figure 3-1: Event Shape Diagrams for "Eat"



Another major problem in translating the checklist procedures into the task representation is understanding the "parsing" NASA did when initially preparing the Flight Data File documents. Much time and effort went into making the FDF quick and easy for the crewmembers to read, but without the proper training and supplementary documents containing abbreviations and standards, they are almost impossible to decipher. In fact, the checklists are often considered *cue cards* rather than being true *instructional* devices. In order to design the task representation accurately, a complete understanding of the Flight Data File is essential. (See Figure 3-2 for a sample cue card). This report will go into as much detail as necessary so that the conventions used by NASA can be understood.

Figure 3-2: OMS/RCS Post Burn Configuration

```
P  07     AFT L,R RCS
           /He PRESS (four) - OP  (tb-OP)
           /TK ISOL  (six)  - GPC (tb-OP)
           /XFEED    (four) - GPC (tb-CL)

   03     RCS/OMS PRESS sel     - RCS as reqd
                  PRPLT QTY sel - RCS as reqd

   08     L,R OMS
            He PRESS/VAP ISOL (four) - CL
           /TK ISOL (four) - OP (tb-OP)

            * If OMS PRPLT FAIL,              *
            *  /affected TK ISOL (two) - CL *
```

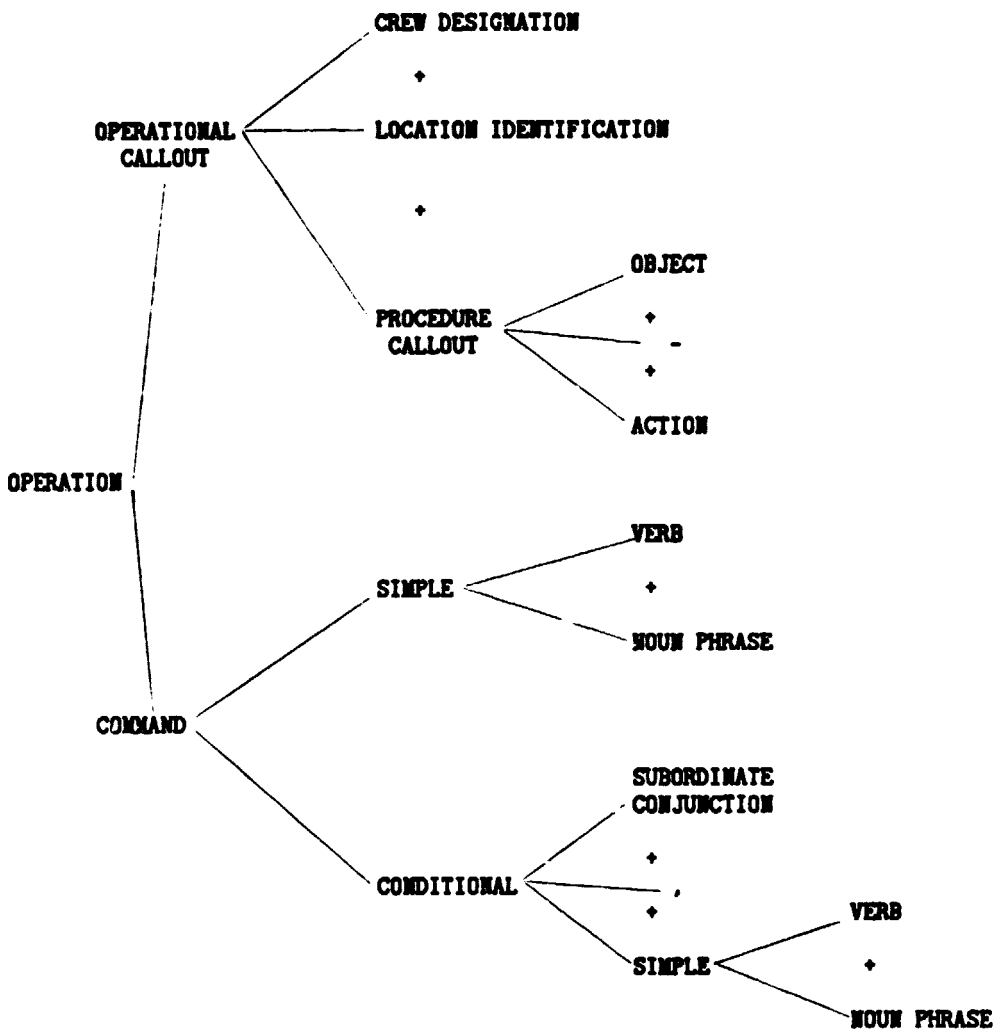
3.2. Checklist Procedures

There are two basic types of operations in the NASA checklist procedures: the *commands* and the *operational callouts*. Figure 3-3 shows how each type is broken down into (or formed from) its components. From our point of view, however, the checklist procedures will only be broken down or translated, never formed. Commands are basically in the imperative form and the operational callouts follow the strict format specified in Space Shuttle Flight Data File Preparation Standards [5]. In both cases the articles "a," "an," and "the" have been omitted for brevity.

The first method of expressing an operation is the operational callout. It is used to describe functions that are associated with displays and controls. The operational callouts that define crew operations are comprised of steps, crewmember designation, location identification, procedure, and notes. Only the procedure is required at all times and it consists of specifying the object to be operated on and the action, the two separated by a dash. The other components are only included as required.

Consider the following examples:

Figure 3-3: Switch Operation Sequence Structure




```
P     R2    1   HYD MA'  PUMP PRESS 1 - NORM
                                    3 - LO
C     C3    2  MAIN ENG LIMIT SHUT DN - AUTO
P     R2    3   HYD MAIN PUMP PRESS 3 - NORM.
```

The far left column contains the crewmember designation. There are two types, general and specific. General crew designations are associated with procedures in which all crewmembers have received equal training and each is capable of performing the task. Some examples of general designations are:

CM1 - first crewman

CM2 - second crewman

SUB - subject

OBS - observer.

Specific crew designations are used in cases where certain crewmembers receive specialized training in a given task. The possible designations are:

C - Commander

P - Pilot

EV1- EVA crewman 1 (extra-vehicular activity)

MS - Mission Specialist

PS - Payload Specialist.

The first two lines are to be performed by the pilot in the above example even though it is only specified in the first line. Only when a new crewmember is to perform an operation is his designation specified.

Location identification is used to specify the location of a particular operation. Identification may be specific, such as a panel or stowage locker number, or general, such as flight deck, middeck, or airlock. A location is usually specified for each step, but in some cases this is unnecessary because there is no need to perform the operation in a particular location. The location is specified in a column to the left of the step number. One or more spaces are always left between the location designation and the step number or the first character of the procedure (when step number is not indicated). In the above example the second line contains no location identification. In such cases the convention is that the location (crewmember and object) remain the same from the previous operation. However, the information must be included for step 3 because of the intervening step 2 that has a different location (crewmember and object).

Panel locations are often combined to save space and to consolidate operations into single steps. For example,

F6 ADI ATT - INTRL

F8 ADI ATT - INTRL

has been combined as follows

F6/F8 ADI ATT - INTRL.

Both statements must be present in the task representation and must be expressed as

separate callouts. This will be discussed in greater detail later when the actual representation is being described. Alternate panel locations may be indicated by using parentheses to indicate that the step is performed at either one location or the other:

F6(F8) ADI ATT - INTRL.

The final portion of the operational callout is the procedural callout, which consists of an object, then a dash, and an action. Most procedural statements refer to switches, valves and frequently used loose equipment. In order to understand these statements the use of upper and lower case letters, spacing, symbols, and special identifiers must be studied. This style of writing was adopted to aid the crewmembers in quickly identifying and accomplishing tasks. Some examples of switch operations are shown in Figure 3-4. It should be noted that the type of switch, quantity, and clarifying comment are not always required. Valve operations appear the same as switch operations except type of switch is replaced by "vlv" standing for valve. There are similar forms for other devices.

Figure 3-4: Switch Operations

PANEL NUMBER	SWITCH TITLE	type of switch	(quantity)	dash	POSITION or ACTION DESIRED	(clarifying comment)
F6	HSI SEL MODE			-	ENTRY	
C3	AIR DATA PROBE		(two)	-	STOW	
	SRB SEP	pb		-	SEP	
C2	TIMER SET	tw	(four)	-	ON	

To accurately represent the operations as described on the cue cards, the conventions used to conserve space and simplify them must be accounted for. One such method is combining switch callouts. Special attention must be given to properly identifying the number of switches being positioned and the particular switches desired from a given group. Any series of callouts such as

BOILER CNTLR 1 - ON

BOILER CNTLR 2 - ON

BOILER CNTLR 3 - ON

will be condensed into

BOILER CNTLR (three) - ON.

To turn on only two of the three selected controllers, the choices are separated by commas:

```
BOILER CNTLR 1,3 - ON.
```

To identify alternatives, the choices are enclosed in parentheses:

```
BOILER CNTLR 1(2,3) - ON.
```

The task representation must interpret these combined operations as multiple tasks and return the condensed version back into its original expanded form.

Another simplifying convention for operational callouts that must be expanded is recreating the deleted leading redundant words, panel numbers, and row numbers. For example,

```
08   LTG PNL R OVHD - OFF
              R INST - OFF
```

must be represented as representation, as:

```
08   LTG PNL R OVHD - OFF
08   LTG PNL R INST - OFF.
```

The checklist operations often require clarification or identification of special conditions. Three different forms of information may appear in a procedure: general notes, cautions, and warnings. All three are written in narrative form and precede the operations to which they apply. The text of each is indented at least two spaces from the margin in order to set the information off more readily from the body of the procedure. Cautions are "boxed" once; warnings, twice (double line box). Although valuable information to the crewmembers appears in the procedural notes it will not be used in the task representation. More likely they will be passed as text to appear as captions in the graphical simulation.

The checklist command is used primarily for operations not associated with displays or controls and may be of two types: simple and conditional (see Figure 3-3). The simple command has normal imperative structure beginning with a verb. For example, a procedural operation telling a crewmember to install film into a camera is expressed as

```
Insert film into camera.
```

The conditional command also reflects basic imperative sentence structure except that

the command is preceded by a conditional clause introduced by a subordinating conjunction such as "if" or "when." For example, a procedural operation telling a crewmember to advance film after finishing some exposures would be stated:

When exposures complete, advance film.

In the case of the conditional commands, Allen's temporals would play an important role in translating the command into the task representation.

Figure 3-5: Sample NASA Checklist with Commands and Operational Callouts

ACES

1 Setup

MA16N	Unstow M-GAMS, CDTR, and CDTR Cable
OC	Attach M-GAMS and CDTR to bracket
MO13Q	√DC UTIL PWR MNB - OFF
MO52J	√DC UTIL PWR MNA - OFF

CAUTION
AEM and MLR share MO52J

MO13Q	Attach OC Pwr Cable to DC UTIL PWR MNB outlet
MO52J	Attach EC Pwr Cable to DC UTIL PWR MNA outlet adapter box

2 Activation

CDTR	Engage Capstan - push button, slide Capstan fwd
M-GAMS	ACCEL PWR - EXT
CDTR	Visually check tape motion
	Rcd times - CDTR ___:___ MET ___:___

Various FDF documents contain the checklist procedures for on-board experiments.

These describe the type of actions to be performed in the environments that are relevant to the TEMPUS project simulations (see Figure 3-5 for an example). Most of the statements are the command type but both types are frequently mixed. The parser for the task representation will have to be able to determine which type of operation it is line by line and parse into identical representations, irrespective of its original form (operational callout or command). When a operation is in the command form it sometimes appears similar to the callout. For example,

```
HATCH     EQUALIZATION vlvs (two)  - NORM

          Close and Latch Hatch

R11P      Remove and stow Inner Hatch.
```

Examples have been found, however, that do not follow the protocol of the procedural callout (see Figure 3-6). These will require more general parsing strategies discussed in Section 4.

Figure 3-6: Sunshield Manual Opening

```
EV:  1.  Restrain inboard aft sunshield cable
         w/tethered vise grips on stbd side

     2.  Cut inboard aft cable w/EMU scissors

     3.  After cutting, restrain cable w/tape

     4.  Restrain outboard fwd sunshield cable
         w/tethered vise grips on stbd side

     5.  Cut outboard fwd cable  w/EMU scissors

     6.  After cutting, restrain cable w/tape

     7.  To open sunshield, pull between uncut
         cables using stbd handrail for restraint

     8.  Restrain sunshield using adjustable wrist
         tethers
```

3.3. Task Representation

The *parser* translates the checklist procedures into a target structure called the task representation. This structure is in turn passed to the simulation system and TEMPUS (see Section 5). The proposed representation is taken from Schank's case frames and primitive verbs. The task frame needs at least six relevant slots:

1. *NUMBER*: an ordered number for each command frame -- the number will provide an ordering of the command and will be used in describing the temporal relationships between the events. This number will most likely be different from the step number in the operational callout.

2. *AGENT*: this is the actor in the command (crewmember designation), and may not have been specified in the callout. In the case that there is no crew designation (and one cannot be arrived at) a blank will be passed. The simulation will assign a crewmember (depending on who is available, who can get there quickest and easiest, and what type of action it is (for example, may be restricted to pilot).

3. *OBJECT*: a specific object or list of objects with OR connectors. Switches, valves, latches, and loose equipment are some examples of possible objects.

4. *ACTION*: the action or position desired as specified by the callout. Some examples are ON, OFF, OPEN, Stow and Rotate.

5. *TEMPORALS*: following Allen's 13 temporal relationships. The objects of the temporals will be the NUMBER's of the other pertinent command frames or the word TIME refering to the contents of the TIME slot.

6. *TIME*: the amount of time to carry out a particular action, for example push and hold for 2 seconds. Can also be used for "after 5 minutes" in combination with the temporal AFTER.

In order to fill the ACTION slot from an operation in the form of an operational callout (not a command), the position desired for that particular switch or valve will be placed in the ACTION slot as it appears in the callout (the word following the dash). To fill the ACTION slot from a command, a primitive verb or list of primitive verbs will be used (Section 4).

A list of verbs that might possibly appear in a callout has been compiled from the samples of documents sent by NASA (Table 3-3). From this list of verbs a list of primitive verbs can be derived. The action verb will be passed in the slot and the primitive verb or list of primitives will replace it later.

The reasons for using the primitive verbs to represent the actions of the command type and not the operational callout type are as follows. First, the list of possible verbs for the commands is very long and each verb may require a series of simple actions. Second, the possible positions for a switch or valve is limited and that type of information will be kept in an attribute table for each switch or valve. The importance of an attribute table is to help understand the verb and determine the physical action required for the simulation. In Section 4.4.2 we discuss the form of the object attribute table frames.

Table 3-3: A List of Action Verbs

activate	dispose	hook	prep	slide
adjust	doff	increase	press	snap
align	don	initialize	pull	stain
assemble	draw	inject	push	step
assist	drop	insert	raise	stow
attach	egress	install	reconfigure	swab
backout	engage	insure	record	swap
balance	ensure	lace	reengage	switch
become	enter	latch	release	take
bite	evaluate	lift	remate	tape
calibrate	exchange	load	remove	tense
centrifuge	execute	locate	repeat	tighten
check	extend	lock	replace	track
clamp	extract	log	report	transfer
close	fill	loosen	reposition	turn
collect	find	lower	restow	turnoff
complete	fix	mark	restrain	twist
config	focus	match	retorque	unfold
connect	fold	mount	return	unhook
consult	gel	move	rotate	unlatch
contact	generate	note	run	unscrew
copy	get	notify	screw	unsnap
cough	give	observe	scrub	unstow
cut	goto	open	secure	unwind
decrease	grab	operate	select	verify
descend	grasp	peelback	set	zoom out
detach	handloosen	perform	setup	
disconnect	handtighten	place	shake	
disengage	hold	position	show	

The primitives will be words like *GRASP*, *ROTATE*, and *MOVE*. The verb as it appears will be put into the ACTION slot, and will be translated into a primitive verb or list of primitive verbs later on. This will simplify the process and keep the ACTION slot relatively small. The idea is that there will be attributes for different objects so that *ROTATE* could mean *wind, screw, turn, twist, loosen, etc.* depending on the object. The differences would be in manner, degree of difficulty, and scale. Verbs like *unscrew, unwind,* and *retorque* differ only in the direction of the rotation from their positive counterparts. A verb like *lift* could be described as *GRASP* and *MOVE*. Some more complicated verbs like *stow, secure, locate,* and *setup* will require a longer list of primitive verbs than *push, pull, press, latch,* and *hook*.

To further illustrate the parsing technique to translate the operation into the task representation, Figure 3-7 shows an example cue card from an FDF Orbit Pocket Checklist. Missing from this cue card are the crew designation and step numbers. The step numbers are relatively unimportant since a number will be assigned to each "event"

Figure 3-7: Example of Checklist Prior to Parsing

```
BUS LOSS ACTION
MNC ALC3

   R2   BLR CNTRL PWR/HTR 2 - B (1 - B on 102)
                          3 - A
   L1   FLASH EVAP CNTRL PRI A - OFF
                         PRI B - ON
        TOP EVAP HTR NOZ R     - A AUTO
   A12  APU HTR GAS GEN/FUEL PUMP 2 - A AUTO
                                  3 - B AUTO
                TK/FU LINE/H2O SYS 2A,3B (two) - AUTO
```

as it is translated. This example is rather straightforward as far as temporals are concerned, therefore the operations will be performed in the order that they appear. The crewmember(s) chosen to perform the operations will depend upon the context in which this cue card is used. One possibility is that it is called from another operation. In that case whoever is doing that instruction will carry out all of the operations on this cue card.

The first column that is indicated in this example is the location identification. It should be obvious that the lines that do not have a location specified occur at the same location as the line above. Other redundant information (switch titles and parts of switch titles) have also been omitted. The task representation must contain as much information as possible about each operation. Omitted information must be extrapolated back into the command frame.

In every line, the group of capital letters and all upper case words following the dash (-) will be put in the ACTION slot. The location and the title of the switch will be put in the OBJECT slot. No temporals are involved in this example.

Extrapolating all the hidden information in Figure 3-7 will lead to the expansion of the callout as shown in Figure 3-8. This expansion can then be parsed into the command frame suitable for input to the simulation. Figure 3-9 shows the parse in tabular form.

3.4. Variations on a Theme

There are numerous variations encountered in examples of cue cards that are not as straightforward and involve some additional rules to parse correctly, for example:

```
P     07    TK ISOL (six) - GPC
```

Figure 3-8: Expanded Checklist of Figure 3-7.

```
R2   BLR CNTRL PWR/HTR 2 - B (1 - B on 102)
R2   BLR CNTRL PWR/HTR 3 - A
L1   FLASH EVAP CNTRL PRI A - OFF
L1   FLASH EVAP CNTRL PRI B - ON
L1   TOP EVAP HTR NOZ R     - A AUTO
A12  APU HTR GAS GEN/FUEL PUMP 2 - A AUTO
A12  APU HTR GAS GEN/FUEL PUMP 3 - B AUTO
A12  APU HTR TK/FU LINE/H2O SYS 2A,3B (two) - AUTO
```

The last line can be expanded further to

```
A12  APU HTR TK/FU LINE/H2O SYS 2A  - AUTO
A12  APU HTR TK/FU LINE/H2O SYS 3B  - AUTO
```

Figure 3-9: Representation of Figure 3-7.

NUMBER	AGENT	OBJECT	ACTION	TEMPORAL	TIME
1		R2 BLR CNTRL PWR/HTR 2	B		
2		R2 BLR CNTRL PWR/HTR 3	A		
3		L1 FLASH EVAP CNTRL PRI A	OFF		
4		L1 FLASH EVAP CNTRL PRI B	ON		
5		L1 TOP EVAP HTR NOZ R	A AUTO		
6		A12 APU HTR GAS GEN/FUEL PUMP 2	A AUTO		
7		A12 APU HTR GAS GEN/FUEL PUMP 3	B AUTO		
8		A12 APU HTR TK/FL LINE/H2O SYS 2A	AUTO		
9		A12 APU HTR TK/FL LINE/H2O SYS 3B	AUTO		

must be extended and represented in the frame:

```
P    07   TK ISOL 1 - GPC
P    07   TK ISOL 2 - GPC
P    07   TK ISOL 3 - GPC
P    07   TK ISOL 4 - GPC
P    07   TK ISOL 5 - GPC
P    07   TK ISOL 6 - GPC.
```

It is absolutely necessary that the task representation appears exactly like this, or else the simulation will not perform the correct sequence of actions.

Sometimes two locations are expressed in the same command using either a conjunction or a disjunction. The following two examples illustrate this:

```
F6/F8  FLT CNTRL PWR - OFF

F6(F8) FLT CNTRL PWR - OFF.
```

For the first example, the *conjunction*, two separate commands must be represented individually as

F6 FLT CNTRL PWR - OFF

F8 FLT CNTRL PWR - OFF.

For the *disjunction*, F6(F8), the OBJECT slot will contain the two switch locations and titles separated by the disjunction "OR". This will be parsed further in the simulation stage. One reason for this is that the simulation will know exactly where the crewmembers are. If a crewmember is at the optional location (F8), it would much more efficient to have him perform the action there than to have another member move to the location mentioned first. That is how it would be done in a human environment.

As mentioned earlier, leading redundant locations, words, or switch titles are deleted from the checklist for brevity. The parser must backtrack and fill in the omitted descriptors. If this is not done, or not done correctly there will be inaccuracy and ambiguity in the task representation.

A check (√) proceding a procedural callout means *verify* or *check*. This indicates that the operation may not necessarily have to be done but that the talkback (discrete indicator) must be in the state listed. The simulation will then have to check the attribute table to find out the present state of the object in question. If it is not in the correct state then the operation must be performed. For example,

F4E √ CNDS DUMP ISOL VLV - CL

means, "Check that the condensate tank dump ISOL valve is closed; if not close it." The OBJECT slot will contain the location identification and the valve title. The ACTION slot will contain "CHECK CL".

Bold face titles and enumerated titles need not be translated into the task representation. They are for the crewmembers' use in locating a particular cue card. If desired, the bold face titles can be passed as text to be used as captions on the display screen. An example of enumerated titles appears in Figure 3-10. The enumerated titles are always underlined descriptions of what operation follows and are not necessary for an accurate task representation.

A very important consideration when parsing the checklist procedures is the representation of the temporal relationships between operations. Some simple relationships are seen in the examples in Figure 3-11 which can be translated quite literally into one or several of Allen's temporal relationships. "When pwr restored"

Figure 3-10: Enumerated Titles

```
1  GAS Encdr Unstow/Setup

   Unstow GAS encoder from locker, connect
     cable to SSP (L12U)

2  GAS Relay nn State Change From X to Y

   Enter <CLR> or <CLR> <CLR>
   √ Display: A--A or A.-.-.A
   Enter two digit relay number <nn>
   Enter <O/S>
   √ Display: P nn Y
```

Figure 3-11: Temporal Statements in Checklists

```
   Maintaining unlocking force, rotate lever
     cw to NORMAL position
-------------------------------------------
   Next, thread the tape around the outside of
     Capstan Tape Guide Roller
   Finally, thread the tape around the inside
     of the Capstan. . .
   After the tape is threaded, wrap the free
     end around the takeup reel. . .
   While holding the tape securely, rotate
     the reel counterclockwise until the
     tape . . .
   Continue winding until the tape is taut
-------------------------------------------
   When pwr restored, cmd relay 27 - L to H
-------------------------------------------
   Before mounting extensions,
     √ PWR TOOL - OFF
```

would use the temporal primitive AFTER. "Maintaining" would call for DURING, the same for "while." This is all straightforward translation. The translation of Figure 3-12 would not as easy to visualize. This example involves parallel sequences of actions in more than one instance. The interpretation of the Egress Test would be that of Figure 3-13.

There is a question of whether or not AFTER is necessary when the numbered procedures occur in perfect sequence with no overlap. It should be obvious that 12 is AFTER 11 and 13 is AFTER 12 in all cases unless specified. It is conceivable that there could be a different interpretation; however, it is assumed that the most straightforward one is to be used. It is therefore proposed that the temporal AFTER not appear in the

Figure 3-12: Egress Test

```
                Crewmen assume places:
                 MS1 in AFT flight deck
                 MS2 at SAL
                 PS1 at Rack 12 foot restraints
                 PS2 at Rack  9 foot restraints

MS1             Announce start of test

MS2             During egress,
     F6D         CAB DEPRESS VLV - op

                After last crewman egresses,
                 close and latch hatch

MS1             Log end of test

                After egress test, open hatch per decal
     F6D         CAB DEPRESS VLV - cl
```

Figure 3-13: Task Representation of Figure 3-12.

NUMBER	AGENT	OBJECT	ACTION	TEMPORAL	TIME
1	MS1	AFT Flight deck	GOTO	EQUAL	2 3 4
2	MS2	SAL	GOTO	EQUAL	1 3 4
3	PS1	Rack 12 foot restraint	GOTO	EQUAL	1 2 4
4	PS2	Rack 9 foot restraint	GOTO	EQUAL	1 2 3
5	MS1	Start of test	Announce	AFTER	1 2 3 4
6	MS2	F6D CAB DEPRESS VLV	OPEN	DURING	7 8 9 10
7	MS1		Egress	DURING	8 9 10
8	MS2		Egress	DURING	7 9 10
9	PS1		Egress	DURING	7 8 10
10	PS2		Egress	DURING	7 8 9
11	MS2	Hatch	CLOSE and LATCH	AFTER	7 8 9 10
12	MS1	End of test	Log	AFTER	11
13	MS1	Hatch	OPEN	AFTER	12
14	MS1	F6D CAB DEPRESS VLV	CLOSE	AFTER	13

representation of sequential, non-overlapping operations (NUMBER's 12, 13 and 14 in Figure 3-13).

The final example of expressing temporal relationships correctly is the case where a specific amount of time is required.

Wait 5 mins. or After 5 mins.

In either case the correct temporal primitive should be chosen and the word TIME should be placed next to it in the TEMPORAL slot. The correct required amount of time will then be placed in the TIME slot.

TEMPORAL	TIME
AFTER TIME	5 mins.

4. Processing Commands

Commands will be translated into the same case frame as the operational callouts. The commands are in the imperative form, or a modified imperative form. The command type operations will require a limited natural language processor. It will have to be able to recognize the first word of the command as either an action verb or some conditional or temporal. From there it will translate the command depending on the first word as either simple or conditional command. This is where the action verb list becomes extremely useful. A list of possible temporals will also be necessary, and can be compiled after more examples are studied. In the case of the conditional command, the subordinate clause is followed by a comma and the remainder of the command will be translated as a simple command. In most cases the articles "a," "an," and "the" have been deleted from the operation. If any one of the articles is found, it need not be translated, and may be discarded.

4.1. A Bottom-Up Parser

A parser is needed to convert the command or sentence into a standard computational structure that a program can use. Using a *bottom-up parser* (BUP) [6] (adapted from Franz to Common Lisp by Gangel), many of the common command phrases that may be used in a specific environment can be easily parsed. BUP is a bottom-up parser for augmented phrase-structured grammars. To explain these concepts, we start with the notion of bottom-up parser. Bottom-up and top-down parsing, although often yielding the same result, vary greatly in their strategies.

Top-down parsing begins at the top (the top rule, often labeled with the symbol S) and tries other rules that may eventually lead to the constituents (grammatical units) of the sentence. For example, the sentence "Jeff turned the valve" would be top-down parsed in an order similar to:

```
S   -> NP VP
NP  -> N
N   -> Jeff
VP  -> V NP
V   -> turned
NP  -> DET N
DET -> the
N   -> valve.
```

The result of the parse is:

```
[S
  [NP [N Jeff]]
  [VP
     [V turned]
     [NP [DET the] [N valve]]]].
```

(In this Lisp-like list notation it may be seen that the pairs of items in braces are associated as an attribute-value pair: thus **[N Jeff]** implies **Jeff** is a **N** (noun); **[NP [N Jeff]]** implies **[N Jeff]** is a **NP** (noun phrase); *etc.*)

Bottom-up parsing begins with the words of the sentence and looks for rules whose right-hand sides match constituents of the sentence. The left-hand side of each rule then forms new constituents which are again matched, and so on, until it becomes a single left-hand side (the symbol S). Using the same example, it would be bottom-up parsed in an order such as:

```
N   -> Jeff
NP  -> N
V   -> turned
DET -> the
N   -> valve
NP  -> DET N
VP  -> V NP
S   -> NP VP.
```

The resulting structure is identical to that of the top-down parse:

```
[S
  [NP [N Jeff]]
  [VP
     [V turned]
     [NP [DET the] [N valve]]]].
```

BUP follows an order similar to what appears in the above example. However, we must still define the notion of *augmented phrase-structured grammars*. An augmented phrase-structured grammar allows an augmentation to the rules in the parser to provide additional information useful for parsing. For example, *Jeff* is an animate object and *the valve* is an inanimate object. Without these augmentations, the sentence, "The valve turned Jeff" could be parsed by the given rules. If the verb *turn* requires that the subject must be animate, only "Jeff turned the valve" would be allowed. BUP allows such augmentations to be included in the rules.

BUP also allows various output formats depending on the given rules. The output currently used is in the form of a case frame. However, it is *not* a true case frame. Without delving deeply into the main verb of the sentence, a true case frame cannot be formed. Although a partial semantic and syntactic analysis is made, there are still certain syntactical and semantical rules that have not been followed.

It may seem that more of the analysis (especially the syntax) should be done in the parser. We opted, however, to separate the deeper analysis for a number of reasons, the main reason being that BUP, although very useful, is not the best parser available. (It happens to be available to us.) Were the two analyses not separated, we would have to depend on only one parser, BUP, without an easy change to another. In addition, if the sentence is at least partially correct, then there is still a chance that the information can be understood and the true meaning found.

The rest of the analysis is now dependent on a verb database containing the verbs that may be used and a list of syntactic and semantic rules dependent on each verb. This analysis forms a true case frame. The output from BUP, using certain rules, is what we call a *partial case frame.*

4.2. Partial Case Frames

As we discussed in Section 3.1, case structures are used to represent the meaning of a sentence independent of its particular surface form. Thus the sentences

```
Jeff turned the valve
```

and

```
The valve was turned by Jeff
```

both have the same deep structure. In Section 3.1 we saw that Schank's CD (Figure 3-2) extends the case grammar by reducing all verbs into a set of limited primitive verbs. As a result, both of the above sentences would be reduced to the same case frame:

```
EVENT
   actor:  Jeff
   action: PROPEL
   object: the valve.
```

Case frames contain a set of roles and the constituents that fill them. The roles in the above example, ACTOR, ACTION, and OBJECT are filled with their constituents, *Jeff*, *PROPEL*, and *the valve*. BUP, although it only produces a partial case frame, shows the same type of output:

```
((OBJECT Jeff)
 (ACTION turned)
 (OBJECT (the valve))).
```

The partial case frame has not yet, however, deeply analyzed the verb *turn*. As will be seen later, once the verb is analyzed and the frame's roles are filled, the verb may also reduced to a primitive.

Continuing with the case frame concept, every case frame contains a list of roles, each of which is dependent upon the frame's verb. Roles, such as

```
AGENT:       the instigator of an action
OBJECT:      an obligatory case found or implied with
             each verb
INSTRUMENT:  the object used to perform the action
LOCATIVE:    it specifies the location or direction
             (source - destination) of the object
STATE:       it specifies the state (initial - final)
             of the object
TIME:        the time at which the action took place
```

allow the concepts within a sentence to be classified into their contextual meanings. (The roles listed for the operational callout task representation may also be present, but we shall not need to refer to them further here.)

For example, let the object, VLV, be a rotary valve with six states labeled 1, 2, 3, 4, 5, and 6:

```
   3  4
2---   5
   1  6
```

The sentence, "Ask the commander to turn VLV from 2 to 5" would be parsed by BUP into the partial case frame,

```
((ACTION ask)
 (ACTOR you)
 (OBJECT (the commander))
 (INSTRUMENT
     (ACTION turn)
     (ACTOR (the commander))
     (OBJECT (vlv))
     (STATE (INITIAL 2))
     (STATE (FINAL 5))))
```

where, according to a transformational grammar rule entitled *object-controlled equi* reformats the sentence into two sub-statements, "Ask the commander" and "The commander to turn VLV from 2 to 5." They are then changed into a partial case frame where, according to the syntactic and semantic features of the constituents, certain roles are filled by the constituents of the sentence.

The sentence "Ask the commander to turn VLV to the right," although similar to the previous sentence in that state 5 is "to the right" of its present state, would be parsed into a different case frame:

```
((ACTION ask)
 (ACTOR you)
 (OBJECT (the commander))
 (INSTRUMENT
     (ACTION turn)
     (ACTOR (the commander))
     (OBJECT (vlv))
     (DIRECTION (DESTINATION (the right)))).
```

The first sentence provided a full account of what action is to be done (in this case by the simulation). The simulator knows (or eventually will know once the case frame is reduced to a suitable message format for the simulator) the state change of VLV, and as long as the action is viable, will be able to perform the event. On the other hand, the second sentence only provides general locative information. "To the right" usually does not provide enough information for a simulation. If VLV is not an *endless* valve (*i.e.* it cannot be rotated from 6 to 1), then "to the right" can account for any of the four remaining states "to the right" of state 2. If VLV is *endless* then "to the right" can account for all of the states[2] except the present state 2. Thus, although the partial case frame contains the information stated in the sentence, there is often a certain amount of information that is still needed for simulation. Therefore, the second case frame is incomplete. It is partially structured, but the role values pertaining to the verb are not complete. Any required roles must be filled and an error analysis must be performed. This all depends upon the verb.

4.3. Using the Verb to Determine a Case Frame

Every verb has an associated case frame(s). It specifies which roles the verb may take, their preferred ordering, and whether or not any of the roles are syntactically obligatory. For example,

```
GIVE - [ (agent) (benefactive) object ]
KILL - [ (agent) object (instrument) ]
```

where the roles in parentheses are optional and the order listed is the preferred order, are two motion (or action) verbs. They assert a change of state or an activity instigated by an agent. As can be seen, the syntactic roles of the two are different. GIVE, as in the sentence "John gives Mary the book," has a benefactive, Mary. But KILL does not. "John kills Mary the book" is obviously incorrect. KILL needs an object, not a benefactive. Thus "John kills Mary" is correct (structurally, not morally!).

---

[2]In many other cases, "to the right" is specific. If VLV was not *endless* and it's present state was 5, then "to the right" would imply "the state 6." For other objects, such as a toggle switch where only two states exist and movement can only be done in a left-right direction, "to the right" is both valid and specific.

In the TEMPUS environment most of the verbs we see are *motion* verbs. Since this is an interface for a simulation, we want to assert a change of state for the object(s) or a motional activity for the agent(s). Of the lengthy lists of action verbs available (Table 3-3 and see also [8, 2]), the verbs we chose to investigate are PUT, TURN, ROTATE, OPEN, CLOSE, and MOVE, as well as some of their multi-word verb forms, PUT ON, PUT ASIDE, PUT DOWN, PUT OUT, TURN ON, TURN OFF, TURN UP, TURN OUT, TURN OVER, OPEN UP, and CLOSE DOWN. Of the numerous definitions of these verbs, we chose the definitions that are pertinent to the given environment.[3] These verbs, besides being motion verbs, also have *key* case frames which follow the main (most used) definition of the verb (Table 4-1).

Table 4-1: Verb Case Frames

```
PUT        - [ (agent) object locative ]
TURN       - [ (agent) object (locative) (state) ]
ROTATE     - [ (agent) object (locative) (state) ]
OPEN       - [ (agent) object (locative) (state) ]
CLOSE      - [ (agent) object (locative) (state) ]
MOVE       - [ (agent) object (locative) (state) ]
PUT ON     - [ (agent) object ]
PUT ASIDE  - [ (agent) object ]
PUT DOWN   - [ (agent) object (locative) ]
PUT OUT    - [ (agent) object ]
TURN ON    - [ (agent) object ]
TURN OFF   - [ (agent) object ]
TURN UP    - [ (agent) object (state) ]
TURN DOWN  - [ (agent) object (state) ]
TURN OUT   - [ (agent) object ]
TURN OVER  - [ (agent) object (state) ]
OPEN UP    - [ (agent) object ]
CLOSE DOWN - [ (agent) object ]
```

There are, however, often other case frames for the verb. Many motion verbs, as well as other types of verbs, have instances of polysemy (multiple meanings). Usually, the semantics of an entire sentence (and often an entire concept) is needed to determine the *true* meaning in the given instance. For example, consider the verb TURN. "Turn the valve on" may imply a rotary movement of the valve to the *on* position. "Turn the switch on" may imply a linear movement of the switch to the *on* position. "Turn the light on" may imply that the switch for the light, not the light itself, is "Turned on." In

[3]TURN OFF has such definitions as: A. to stop the flow of (water, gas, etc.), as by closing a faucet or valve, B. to extinguish (a light), C. to drive a vehicle or walk onto (a side road) from the main road, and D. slang: to cause (someone) to lose interest; to bore or discourage. A and B are definitely pertinent. C may be pertinent, but not really in such an enclosed environment as the shuttle. D is definitely not pertinent. Nor is it really valid to use it in a command (eg. "Tell John to turn on"). Thus only A and B would be used.

addition, "Turn out the light" is a valid sentence. But, "Turn out the switch" or "Turn out the valve" are not valid sentences. The verb TURN OUT is only valid for an indicator.

As a result, the meaning of a statement can often be found only after the verb and its roles have been analyzed. BUP changes the sentence into the partial case frame. The next step is to analyze the partial frame into its true case frame. Both the verb and the *filled* case roles plus any default information from former discourse that may be pertinent to the present statement must now be used to fill or correct the true case frame.

4.4. Analysis of the Partial Case Frame

In the following sections we try to determine which roles are pertinent to the verb's meaning. We will examine the verbs themselves, the objects to which they refer, variations in sentences, and disambiguation by questions directed to the user.

4.4.1. A Verb's Meaning

BUP's output is a partial case frame dependent on an action verb. As seen earlier, each case frame contains certain syntactical information that is pertinent. For example, consider the verb PUT. PUT contains a syntactical form:

```
[ (agent) object locative ].
```

The sentence, "John put the book on the table," fills all of the case roles, both optional and obligatory. *John* is the agent, *the book* is the object, and *the table* is the locative. Since the agent role is optional (has parentheses), the sentence "Put the book on the table" is also valid. It would be allowed because we have structured the rules in BUP that way. However, if the user neglected to include an object role, the rules would not allow the parsing. PUT is a transitive verb and cannot use the agent as the object[4].

The sentence, "John put the number in the log," again fills all the required roles. *John* is the agent, *the number* is the object, and *the log* is the locative. However, the meaning of PUT in this context is different. *The number* is not a physical object. Instead of John physically placing *the number* in the log, John must now write *the value of the number* in the log. The same verb and the same case frame are used, but a totally different underlying meaning exists. The realization that the object is not physical plus

[4]Turn is both a transitive and intransitive verb. As such, "John turns," an intransitive form, implies the fact that *John* is both the agent and object. The sentence is actually, "John turns John."

the fact that information can be written into a log can be used to determine its true definition.

Finally, the sentence "John put out the light" shows another definition for PUT. PUT OUT is a multi-word verb composed of the verb *put* and the modifier *out*. If the sentence was "Put out the garbage," then the physical movement of "the garbage" would follow PUT's meaning. But in "put out the light," it must be understood that *the light* implies a physical object which emits light. Next, the physical movement of the light emitter would usually not follow the statement's underlying meaning (unless tightening a light bulb would allow an electrical connection). In this case, specific knowledge about the object which controls the light emitter is needed.

A lot of underlying information is needed to determine a sentence's true meaning. A list of the verb's meanings, the context of the sentence and its surroundings both from previous statements and the overall environment, and any given or implied information all help toward determining the meaning of the verb.

4.4.2. The Object's Role

As seen earlier, TURN has the syntactical definition,

```
[ (agent) object (locative) (state) ].
```

The roles, agent, locative, and state are all optional. However, although the user may omit those roles, the information is still needed. If an agent was previously mentioned, then that agent is implied. The two other roles are both dependent on the object. To clarify this problem, we must introduce the system's *object frame*.

The object frame is a structure with slots and values which describes needed detail about an object's structure, states, motions, and so on. All the "knowledge" needed to operate, move, or change an object's state is embedded in the values stored in the object frame. Presently the object frame has the following structure:

- Name or number of the object
- Texture mapping
  - a texture map pointer or a pointer to and from the object's SurfsUP description
  - object center coordinates
  - object extent box

- Message passer
  - o accepts and sends messages that are specific to the object
- Type of object

```
Control, Indicator, Tool, Clothing, Stowage,
Stowage-Restraint, CRT, Agent, etc.
```

- If Object = CONTROL
  - o Type of control,

```
Switch, Circuit-Breaker (CB),
Valve, Latch, etc.
```

  - o Sub-type of control,

```
Switch: push-button, push-button-with-light,
        thumbwheel, valve,
        display-select-switch, toggle
Cb:     toggle, push/pull
Valve:  lever, rotary
Latch:
  etc.
```

  - o The panel on which it is mounted
  - o The possible directions for the control's state-change

```
Left-to-Right
Up-to-Down
Release-to-Push
Push-to-Pull
Close-to-Open
Counterclockwise-to-Clockwise
Joystick
```

  - o Ordered labeling of the control's states
  - o Amount of movement from one state to the next state, or *vice versa*
    - discrete or continuous range
    - type of measurement
    - an account for each state and the next

```
including:
   the amount of travel
       needed to reach the next state
   the amount of force
       needed to enact the movement
```

  - o Current state (variable)
- If Object = INDICATOR
  - o Type of indicator

```
meter, discrete-event, tones
```

  - o Sub-type of indicator,

```
Meter:          bar, digital, rotary
Discrete-Event: light, off flag,
                talkback-gray, talkback-legend,
                talkback-barber-pole
Tones:
```

- o The panel on which it is mounted
- o Ordered states (to follow the ordering of the linked control)
- o Connections to a control
  containing a list of the pairs:
  an object
  the state it must be in (or nil)
- o Current state (variable)
  color, meter-position, or sound

• If Object = TOOL

- o For now, consider it as a "solid" object
- o Is it static or dynamic? (*i.e.*, can it be moved?)
- o Miscellaneous slots
  including:
  the tools's use,
  the area of the tool to be held,
  preferred grasp configuration,

• If Object = CLOTHING

- o For now, consider it as a "solid" object
- o Miscellaneous slots
  including:
  the clothing's use,
  the clothing's size,
  folding transformations,

• If Object = STOWAGE

- o With or without restraint?
  - If with restraint:
    - o pointer (in SurfsUP) to the stowage-restraint frame
      handle, latches, bungees, etc.
- o Pointer (in SurfsUP) to the objects contained in the stowage
  - the objects would change as they are removed from or replaced into the stowage area
    tools, clothing, etc.

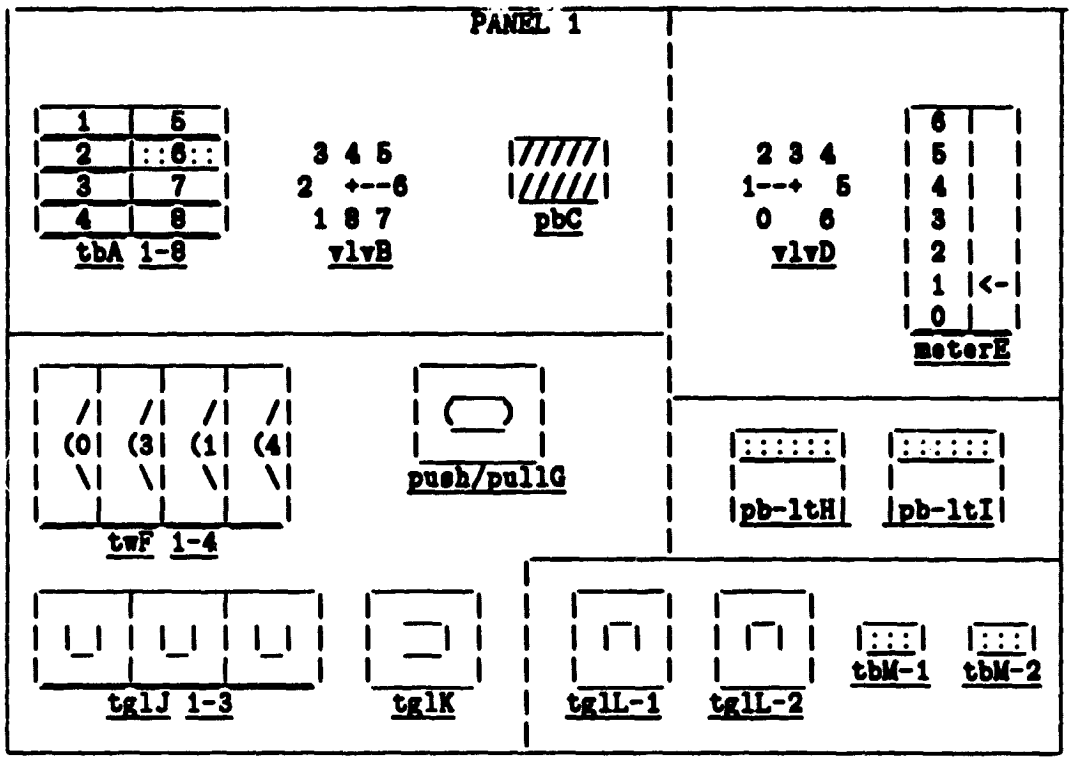
• If Object = STOWAGE-RESTRAINT

```
handle, latches, bungees, etc.
If latch or handle:
    it would be part of SurfsUP's object hierarchy
Else:
    it would be a separate object with a separate
    frame, still accessible from a pointer to SurfsUP
```

- If Object = CRT
  - o viewing center
  - o pointer to SurfsUP geometric description
    - output image on the CRT (perhaps only cosmetic)
- If Object = AGENT
  - o Personal Attributes
    - name
    - nicknames
    - pointer to ADB entry and associated anthropometric parameters
    - role (Pilot, Commander, Mission Specialist, etc.)
    - preferred handedness
    - responsibility (geometric area, panel, etc.)
    - reach model (workspaces generated from TEMPUS)
    - strength model (from specific task or general data)
  - o A Special Message Passer
    - Unlike the other objects, the agent would be intelligent. As such, it would pass and receive messages in a more intelligent, commutative method than the other objects. Besides sending and receiving messages in a straightforward fashion (as in the other objects), the agents can also pass messages to one another to alleviate certain problems that can occur during multi-agent planning for a common goal.

There exists a database of such frames for each object in the environment. Since the present area we are working with involves mounted objects (*i.e.* switches, valves, meters, indicators, *etc.*) on particular panels in the environment, we have created a test panel, aptly called *panel-1* (Figure 4-1) where

Figure 4-1: Hypothetical Panel



tbA are talkback indicators that indicate on or off,
vlvB and vlvD are rotary valves,
pbC is a push button,
meterE is a bar meter,
twF are thumbwheel switches,
push/pullG is a switch that moves in the Z axis,
pb-ltH and pb-ltI are push buttons with an indicator light,
tglJ-1, tglJ-2, tglJ-3, tglK, tglL-1, and tglL-2
are toggle switches, and
tbM-1 and tbM-2 are indicator lights
internally connected to a switch.

Various facts about the objects include:

```
(tbA), (vlvB), and (pbC) are all internally connected.
   To change to another value on (vlvB),
      1. (vlvB) is rotated to a value (eg. 4),
      2. (pbC) is pushed
      3. light 4 on (tbA) lights up
         (and the previous light is off).

(vlvD) and (meterE) are internally connected.
   (vlvD) is not continuous. It only goes from 0 to 6.
   (meterE) displays the position on (vlvD).

(twF) has four separate thumbwheels.
   Each thumbwheel goes from 0 to 9.

(push/pullG) is a push/pull type of circuit breaker.
   Push is ON, Pull is OFF.

(pb-ltH) and (pb-ltI) are push buttons with lights.
   When either is pushed, the light changes
      from its present condition
      to a new condition
      (ie. off to on, or green to red).

(tglJ-1), (tglJ-2), and (tglJ-3) are 3 distinct
   toggle switches that move in an down-up direction.

(tglK) is a toggle switch that moves
   in a left-right direction.

(tglL-1), (tbM-1) and (tglL-2), (tbM-2)
   are internally connected.
   If the left toggle is in the UP position,
      the left talkback is ON;
   If the right toggle is in the UP position,
      the right talkback is ON.
```

To explain the database, we will use the objects **vlvD** and **meterE**. The structure of the control **vlvD** is

```
(control
   :name         VLVD
   :texture-map  ()
   :message      (IF (NOT (EQUAL (CONTROL-CURRENT VLVD)
                                 (INDICATOR-CURRENT METERE)))
                     (SEND '(SET METERE TO
                                 ,(CONTROL-CURRENT VLVD))))
   :type-of      VALVE
   :sub-type     ROT
   :panel        PANEL-1
   :direction    (COUNTERCLOCKWISE CLOCKWISE)
   :states       (0 1 2 3 4 5 6)
   :movement     (DISCRETE   DEGREE
                    ((0 1) 45 5)  ((1 2) 45 5)  ((2 3) 45 5)
                    ((3 4) 45 5)  ((4 5) 45 5)  ((5 6) 45 5))
   :current      1).
```

Some of the slots are obvious. The *name* of the object = **vlvD**, the *type-of* = valve, *etc.*, and the current (state) = 1. What needs to be explained are such roles as *texture-map*, *message*, *direction*, *states*, and *movement*.

The *texture-map* is a mapping into the graphical coordinates of the object on a geometric surface. Typically the surface represents a panel upon which the object is mounted. In the present case it has no value.

The *message*

```
(if (not (equal (control-current vlvD)
                (indicator-current meterE)))
    (send '(set meterE to ,(control-current vlvD))))
```

is a message within the data structure of **vlvD** which sends messages to a pattern matcher within the system. Since **vlvD** and **meterE** are *electrically* connected, there must be some method of equating the position on **vlvD** with the value displayed on **meterE**. Whenever **vlvD** is used, the *if* part of the message is checked. If the state of **vlvD** is NOT equal to the state of **meterE**, such as when **vlvD** is changed to state 3, the message "set **meterE** to 3" is enacted. As if it were wired, the meter changes its value.

The *direction* slot contains a pair of opposite directions. The *states* slot (0 1 2 3 4 5 6) is a list of all the states of the object, **vlvD**. The *movement* slot

```
(DISCRETE   DEGREE
 ((0 1) 45 5)  ((1 2) 45 5)  ((2 3) 45 5)
 ((3 4) 45 5)  ((4 5) 45 5)  ((5 6) 45 5))
```

contains much information. The first constituent, *discrete*, indicates that the valve moves in discrete steps. The opposite would be *continuous*; the movement would have no steps. The second constituent, *degree*, is just what it looks like, the units of measurement. The third constituent is a list of cells, each consisting of states and other information. Look at the second cell, ((1 2) 45 5). The first component, (1 2), shows a possible state change for **vlvD**. The position and therefore the state of **vlvD** can be changed from state 1 to state 2. The order of the numbers is unimportant; **vlvD** can also go from 2 to 1. The second component, 45, shows the number of degrees involved in the state change. If the object had *mm* instead as its second constituent, then it would be 45 mm. Finally, the third component, 5, is just an arbitrary number (at this time) indicating the amount of pressure needed to enact a state change. **VlvD** can change its state from 1 to 3 by taking the path from state 1 to state 2, in the second cell, and 2 to 3 in the third cell. State 1 to 2 requires 45 degrees and state 2 to 3 requires another 45 degrees resulting in a total movement of 90 degrees. When following an increasing path in the states list the direction of the 90 degree movement is clockwise. In going from 3 to 1, the other direction would be counter. In addition, since there is no cell containing the states 6 and 0, the valve is not *endless*. It cannot be moved from state 6 to state 0 without going

through states 5, 4, 3, 2, and 1.

The structure of the indicator **meterE** is

```
(indicator
   :name          METERE
   :texture-map   ()
   :message       ()
   :type-of       METER
   :sub-type      BAR
   :panel         PANEL-1
   :states        (0 1 2 3 4 5 6)
   :connect       ( ((VLVD 0)) ((VLVD 1)) ((VLVD 2)) ((VLVD 3))
                    ((VLVD 4)) ((VLVD 5)) ((VLVD 6)) )
   :current       1)
```

As in the control structure, many of the slots are obvious. Only two things need be explained. Since a change of state in an indicator is due the change of state in another inanimate object, movement is not needed, but the *connect* slot is necessary. To explain its use, look at the list in the state slot: (0 1 2 3 4 5 6) is a list of the seven states that exist in **meterE**, compatible with the number of states within **vlvD**. The *connect* slot also contains seven cells. Look at the fourth cell, ((VLVD 3)). Since it is the fourth cell, it corresponds to the fourth value, 3, in the *states* list. If the input to the interface asked the agent to "turn **meterE** to 3," what it really meant is that the state of the control attached to the meter is to be changed to a state which results in the change of **meterE** to 3. In this particular case the state labels are identical, but that is not always the case. What is important are the positions of the cells within the *connect* list in correlation with the object's *state* list. The cell contains one cell which has two values, **vlvD** and 3. The first says that the object that can change the meter to 3 is **vlvD**. The second says that *vlvD* must be changed to state 3 to perform **meterE**'s state change. Each of the seven cells can contain as many cells (o1 s1), (o2 s2),..., (oN sN) of objects and states as are needed to perform the implied action. This information allows an *implied connection* to take the place of a mechanical connection.

For a real person, such information is not usually stated, since the senses allow him or her to obtain much information directly. Sight allows one to determine the amount of movement. Touch allows one to determine the amount of pressure needed, whether or not the object has discrete or continuous positions, and whether or not the object is *endless*. Language comprehension substitutes implied connections. A simulated person cannot do the same, but by creating a database for each object the system can appear to know the same kind of information.

By using the object database, the system can now determine both the locatives and statives pertaining to an object that may be needed in the case frame. Also, the object database provides insight into the true meaning of the verb.

4.4.3. Variations in a Sentence

Sentences such as

a) "Turn vlvD"
b) "Turn vlvD to the right"
c) "Turn vlvD from the left to the right"
d) "Turn vlvD clockwise"
e) "Turn vlvD from 1"
f) "Turn vlvD to 3"
g) "Turn vlvD from 1 to 3"
h) "Turn meterE to 3"

all have similar meanings. We know that **vlvD** is a rotary valve, so our first verb choice is the most prominent, *rotate*. If it had been a toggle switch, a less prominent meaning, *change position by moving through an arc*, would be correct. If a linear switch, it is actually an improper verb, but if we bend the rules the implication is *change the position*. And, if it had been a push button, the wrong verb was chosen by the user.

Using the rotary valve, **vlvD**, the first sentence (a) is a general statement without any modifiers. No statives and no locatives were stated, so nothing can be implied. All that can be determined it that six possibilities exist; 0, 2, 3, 4, 5, and 6 (since it is already in state 1 and some motion was requested). In (b) "to the right" limits the number of states available to 2 through 6, but that is not much more restrictive than (a). Sentences (c) and (d) also present the same problem as states 2 through 6 are all possibilities. Sentence (e) gives the initial state, but since that can be found in **vlvD**'s database, the result is the same as for (a). Sentences (f) and (g) give the final state, so the information is enough to perform the action. Sentence (h) is the example stated before: by delving into **meterE**'s database, the result is the same as (f).

From some of the earlier examples, some case frames need nothing more than just the object. "Turn out **tbM-1**," where **tbM-1** is an indicator light connected to **tglL-1**, requires no additional information from the user. Likewise, "Turn off **tbM-1**" or "Turn off **tglL-1**" provide the interface with an implied final state. In fact, any object with only two states can be used alone with any of the verbs, OPEN, CLOSE, TURN ON, TURN OFF, OPEN UP, and CLOSE DOWN. The two stated objects (they must be controls) which move in the up-down direction can be used alone with TURN UP or TURN DOWN, plus the other six verbs. Most of the other objects do require some sort

of instantiation.

4.4.4. User Queries

Once the sentence is parsed to a partial case frame and the verb case is determined, the optional roles of the real case frame that are still needed for the simulation must be filled. The parser helped to remove some ambiguities in the sentence (**e.g.** if John is the commander, then if the sentence states *the commander* as the agent, *John* is returned). It also checked state validities for the given object. Other simple things, such as using default values for pronouns, were also done in the parser. These role values were both easily determined and common to most action verbs. The more difficult checks are done in the analyzer.

Going back to the sentences (a) through (f) we analyzed above, we saw that (a), (b), (c), (d), and (e) were ambiguous. The most effective method for disambiguation is to pose a question for the user. Since the general meanings of each sentence have already been determined, a specific query can usually be constructed. Sentences (a) and (e) unfortunately limit the system's friendliness, since there is not much more that can be done other than asking which of all the possible (non-current) states the user wants. Sentences (b), (c), and (d), though, do allow a sense of friendliness. "To the right" and "clockwise" present a limit to the available states. As such, the user is asked about only those after the current one in the states list. Sentences (f), (g), and (h) require no more information as long as the given states are correct.

If the current state of **vlvD** were 5, certain things change. Sentences (b), (c), and (d) no longer need any help from the user. The only state that can follow state 5 is state 6. The analyzer can determine the state role. If the object was **tglK** and its current state was *off* (facing the left), "Turn **tglK**" is enough for the system to realize that it should be moved to the *on* position.

In any case, the answers to user queries will result in a complete case frame for the given verb.

4.5. Problems in the Analysis

Up to now, the main type of object we have been discussing were mounted controls and indicators, all with discrete states. Other type of objects obviously exist. However there are many problems in deciphering sentences about objects which have no discrete

states. The most important problem involves the difficulties in determining an exact movement. For example, the verb TURN, when used with a discrete state object, only allows a limited amount of movement to be possible, from state-to-state. With other objects, there are no set state-to-state movements. The only type of modifiers that can help are directives, such as *around, up, down, etc.*, directives with respect to another object, or graphical texture mapping coordinates. Until the interface system is attached to the simulator, the most that can be done is to create the case frame for the sentence.

Not all freestanding objects have those problems. The sentence, "Put on the space suit," needs no directives. The problem lies in the simulator, not the interface. The only problem the interface may have, and can easily fix, is: which space suit? That can be obtained from a user query.

4.6. Reduction to a Primitive Verb

As stated earlier, there are a great many action verbs. Miller gave a list of 217 action verbs, Badler [2] upped the count to 228 verbs, but both of them stated that the list was incomplete. By adding all the multiple word action verbs, the lists become huge. And counting all the definitions connected to each of the verbs, the lists are overwhelming.

Miller reduced all motion verbs into three concepts: *travels, changes location,* or *comes/goes*, depending on the type of action verb. Schank reduced them to PTRANS, PROPEL, and MOVE. Likewise, the interface reduces the motion verbs pertinent to the environment to the primitives

```
PMove    - the physical movement of an object
           including an agent
PWrite   - self explanatory
PGrasp   - place the agent's hand around an object
PRelease - open the agent's hand.
```

Other pertinent verbs can be classified as a type of:

```
PLook    - look at some object
PWait    - pause for a certain amount of time.
```

The primitives, plus certain elaborations obtained from the original verb's meaning, specify the true meaning of the desired action in a more simplistic manner. From this, the simulation can be done.

Given the sentence "Turn **vlvD** to 3," the system determines the meaning of the verb, analyzes the partial case frame (determining the agent, the initial state, and the

location of **vlvD**), and reduces the verb to a primitive. The case frame for this sentence (assuming the default agent is *John* and the default location is *panel-1*) is

```
( (ACTION    PMove)
  (ACTOR     John)
  (OBJECT    vlvD)
  (LOCATION  panel-1)
  (STATE     (INITIAL 1))
  (STATE     (FINAL 3)) ).
```

From here, the case frame is evaluated and reformated into a suitable message to be sent to the simulator.

4.7. Using the Operational Callouts

As we discussed earlier, the operational callouts such as

```
C  R2 HYD-CIRC-PUMP-1 - ON
```

have no verb. C is the actor, R2 is the panel, HYD-CIRC-PUMP-1 is the object, and ON is the final state. The primitive verb, PMove, is used for the physical movement of an object. All control movements end up being reduced to PMove. Since no user inquiry would be needed for an operational callout (it provides all the information that may be asked), the primitive verb is known immediately. The resulting case frame would be:

```
( (ACTION    PMove)
  (ACTOR     C)
  (OBJECT    hyd-circ-pump-1)
  (LOCATION  r2)
  (STATE     (INITIAL off))
  (STATE     (FINAL on)) ).
```

(The tabular format of Section 3.3 is equivalent to the list form used here.)

A similar case exists for sentences involving indicators. Since the indicator is not the object that is moved, the value of its *connect* slot is used. It provides the control object and the final state for the control but, as in the operational callouts, no verb.

Originally it seemed as if the verb used for the indicator would be valid for its control. But in the sentence, "Turn out **tbM-1**," the verb TURN OUT can only be used with indicators. So the same problem exists as in the operational callouts. The *connect* slot values provide a similar amount of information as in the operational callouts, so the primitive PMove is still used.

5. How to Send Messages to the Simulator

Now that a case frame containing all the required information roles needed for a simulation has been created, the information should be evaluated. When discussing the control structure of **vlvD**, we saw that the total movement needed to go from state 1 to state 3 was 90 degrees in the clockwise direction. Also, the pressure had the value 5. This information is what we will be sending to the simulator.

A partial case frame involving an agent, John, the object, **vlvD**, and the final state, 3, would eventually result in the case frame:

```
((ACTION  ask)
 (ACTOR   you)
 (OBJECT  John)
 (INSTRUMENT
     (ACTION    PMove)
     (ACTOR     John)
     (OBJECT    (vlvD))
     (LOCATION  panel-1)
     (STATE     (INITIAL 2))
     (STATE     (FINAL 5)))).
```

By evaluating the state values of the object, the necessary values can be calculated. To get this information to the simulator, a "standard," fully-expanded message is constructed from the frame.

We are presently evaluating a public-domain message-based simulation system called ROSS [7]. In ROSS a valid message to the simulator describing what is to be done would be:

```
(ask John to PMove vlvD 90 degrees clockwise
     to state 5 with a pressure of 5).
```

From this message, all of the known and determined information about the movement is conveyed to the simulator. Were a different sort of simulator available, such as the hierarchic process abstraction and simulation mechanism proposed in a companion report [3], only the realization of the message need change.

Until the simulator and the interface are connected, some sort of pseudo-simulator will take its place. This mock simulator basically has three functions. The first is a pattern matcher to quickly determine the constituent of the message. The second is the formal change of the state(s) of the object, as if it was done by the agent. The third is an acknowledgement to the user that the meaning of the input was conveyed correctly and the action was completed.

6. Conclusions

One form of input to TEMPUS should be the language that NASA already uses to describe crewmember tasks. The operational callouts and the commands in their standardized forms provide such an input. These task instructions offer the user a highly compressed input form, requiring only interactively typed information or recalled command lists from a database. The advantage is in the sympathy between the system's understanding of the command and the ability which an actual crewmember would apply to performing the task. Consequently, the language interface may help to design and validate *new* procedures and enforce the standards for their structure.

Although the operational callouts and commands cover many of the NASA tasks we have examined, it is clear that there are many task descriptions which lie beyond the analysis process presented here. In particular, the details of how to move from station to station, the parsing and understanding of more arbitrarily-structured natural language statements concerning tasks, and the actions necessary to achieve inherently complex maneuvers (such a donning a space suit) are not directly addressed by this interface enhancement alone. Considerable effort will be needed to integrate these more complex situation specifications with the general (zero-gravity) motion simulation [3]. We have attempted in this report to identify the approximate point where the investment in processing more general language structures will not return a greater benefit than the enhancement of the interactive graphical capabilities for planning general zero-gravity human body motion. Although the computational requirements of both are difficult to assess and compare, we do not see dramatic changes in natural language processing costs in the near future, whereas graphical hardware costs continue to decline while real-time display system capabilities increase.

The major processes needed to accomplish this enhanced user interface are reasonably well known: parsing, case frame filling, and simulation message generation. There will be a significant effort required to actually build the object frame database for a non-trivial application and develop verb case frames for a large class of task-oriented motion verbs. Eventually, an interactive system should be added to permit the simple update of the object and verb database, though improvements in parsing and determination of verb semantics may be made directly in program code. The major research effort will lie in determining the semantics of motion verbs and the proper incorporation of temporal information. In addition, there must be an effective integration of all the pieces so that the interface actually functions properly. Finally, we

must be sensitive to computational efficiency: the execution of a command should be no slower than the manual effort required by TEMPUS to perform the same actions!

The connection of the language interface to the TEMPUS simulation system is a challenge that that has no close parallel in the history of Artificial Intelligence research. There is much related work in natural language processing, expert system design, and knowledge representation that can be borrowed, expanded, or modified to fit this problem. The application to actual human movement control, however, has never been adequately demonstrated. The underlying TEMPUS system appears to make this goal possible.

7. Schedule and Resources

The tasks outlined in the Conclusions could be realized over a four year period if suitable personnel were directed to its implementation. The schedule would, of course, differ if other directions were taken. The approximate timetable for the user interface enhancement effort is given in Table 7-1.

Table 7-1: User Interface Enhancement Schedule

Time Milestone	Task (per staff member)
year 0.5	Parse operational callouts. Complete design of verb and object case frames.
year 1	Parse standard commands. Build experimental frame database.
year 2	Build parser which will complete partial case frames. Integrate output from verb frame with simulation system.
year 3	Expand verb database with semantics of frequently-used action verbs. Incorporate temporal information into parser and task simulator.
year 4	Provide interactive interface to maintain object and verb database. Demonstrate control over multiple interacting agents.

The *time milestone* is the length of time from project inception (not a duration) to the completion of the indicated *tasks*. The tasks are a summary of the work needed to fulfill the system requirements discussed in the Conclusions. Each task refers to *one* graduate research assistant. This is a half time load (20 hours/week). Thus multiple tasks for one time milestone are assumed to proceed in parallel, and a total of two

individuals for four years are required.

The resources required are summarized in Table 7-2. The monetary estimates are based on solely on 1985 University of Pennsylvania rates including employee benefits, tuition, and overhead as applicable. There is no provision for inflation; that may be projected by NASA as necessary.

Table 7-2: User Interface Enhancement Resources

```
2 Graduate Research Assistants for duration of project.....50K/year
Faculty supervision time (10% of academic year).............10K/year
Travel, current expense, duplicating, etc...................34K/year

-------------------------------------------------------------------------------
   Totals:
      Year 1: $94K
      Year 2: $94K
      Year 3: $94K
      Year 4: $94K
-------------------------------------------------------------------------------
```

8. Bibliography

1. James F. Allen. "Towards a General Theory of Action and Time". *Artificial Intelligence 23*, 2 (1984).

2. Norman I. Badler. Temporal Scene Analysis: Conceptual Descriptions Of Object Movements. TR-80, Dept. of Computer Science, University of Toronto, February, 1975.

3. Norman I. Badler, Paul Fishwick, Nina Taft, and Mukul Agrawala. Zero-Gravity Movement Studies. Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1985. (For NAS9-16634).

4. Norman I. Badler, Jon Korein, Craig Meyer, Kamran Manoochehri, Jane Rovins, Jeffrey Beale, and Brian Barr. System Integration Report. Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1985. (For NAS9-16634).

5. Flight Operations Directorate, Operations Division. Space Shuttle Flight Data File Preparation Standards. NASA, Lyndon B. Johnson Space Center, Houston.

6. Timothy Finin. BUP - A Bottom-Up Parser for Augmented Phrase Structured Grammars. Franz Lisp Program.

7. Philip Klahr and William S. Faught. Knowledge-Based Simulation. Proceedings of the first Annual National Conference on Artificial Intelligence, AAAI, August, 1980, pp. 181-183.

8. George A. Miller. English Verbs Of Motion: A Case Study In Semantics And Lexical Memory. In *Coding Processes In Human Memory*, Arthur W. Melton and Edwin Martin, Ed., V.H. Winston & Sons, Washington, D.C., 1972, ch. 14, , pp. 335-372.

9. Roger C. Schank. *Conceptual Information Processing.* Amsterdam, North Holland, 1975.

10. David L. Waltz. Event Shape Diagrams. Proceedings AAAI-82, 1982, pp. 84-87.